Complejidad Computacional
Aritmética de grandes números. Multiplicación por Karatsuba

Yani
www.yaniweb.com

25 de febrero de 2014

Resumen

Se presenta una implementación para computar aritmética de números grandes (mayores a 100 dígitos) y multiplicación de éstos mediante el algoritmo de Karatsuba. La implementación se hizo en lenguaje C. Este documento también está en versión PDF.

Índice de contenido

1 Problema **2**

2 Solución **2**
2.1 Enfoque de solución 2
2.1.1 Representación de números grandes 2
2.2 Codificación de Suma utilizando acarreos 2
2.2.1 Algoritmo Suma utilizando acarreos 3
2.3 Codificación de Producto con algoritmo de Karatsuba 3
2.3.1 Historia 3
2.3.2 Algoritmo de Karatsuba 4
2.3.3 Otras funciones codificadas 5

3 Manual de usuario **6**
3.1 Compilación del programa 6
3.2 Ejecución 6

4 Ejemplos ejecutados **7**
4.1 Ejemplo 1 - Suma 7
4.2 Ejemplo 2 - Suma 8
4.3 Ejemplo 3 - Multiplicación con Karatsuba 9
4.4 Ejemplo 4 - Multiplicación con Karatsuba 9

1 Problema

Aritmética de grandes números: Construir un módulo para el manejo de números enteros grandes (número de dígitos ≥ 100). Represente números grandes como listas de enteros sin signos.

Construya la suma de enteros grandes procediendo según la intuición más elemental, llevando acarreos. Construya el producto de enteros grandes procediendo según el algoritmo de Karatsuba, del tipo "divide y vencerás" que aparece en el libro de Ullman: *"Design and analysis of algorithms"*.

2 Solución

El programa se codificó en lenguaje C, se hicieron las funciones aritméticas solicitadas (suma por método tradicional y producto por algoritmo de Karatsuba) y algunas auxiliares necesarias por las primeras. En esta sección se explica y se referencían algunos nombres de funciones del programa principal ubicado en el archivo `main.c` el cual contiene toda la codificación. Dicho archivo fue entregado en el *tarball* del programa 2. El *tarball* puede descargarse aquí o bien desde mi sitio web.

2.1 Enfoque de solución

Debido a que en las computadoras actuales, un compilador de lenguaje C representa comúnmente a un entero en 32 bits, usando 1 bit para el signo y los restantes 31 para el valor numérico, es posible trabajar con números de hasta 2×10^9 aproximadamente, es decir alrededor de 2 mil millones [1]. Sin embargo, un requerimiento del problema es manejar números grandes de más de 100 dígitos y representar a éstos como listas de números, razón por la cual es imposible resolver el problema usando tipos de datos primitivos o librerías de precisión grande que ofrece cada lenguaje de programación.

2.1.1 Representación de números grandes

Como se explicó arriba, es imposible almacenar números muy grandes en una variable de tipo de dato primitivo en C, por lo que la representación de estos números se hizo mediante arreglos de enteros en base decimal; los números son leídos por el programa en modo caracter y cada uno es almacenado en una posición i del arreglo de enteros comenzando por el dígito menos significativo (esta tarea la realiza la función `void readNumber(int aa[], int *x)`), de tal suerte que las operaciones, validaciones y manipulación de dichos números en el programa se hace sobre arreglos de enteros.

2.2 Codificación de Suma utilizando acarreos

Se codificó la función `void add(int A[], int B[], int C[])` para computar la suma de dos números mayores de 100 dígitos representados en forma de arreglo de enteros. Recibe como entrada los números (*arreglos*) A, B y C, y regresa la suma A+B en C.

La suma de dos números se realiza usando el método tradicional, es decir se suma dígito a dígito cada elemento del mismo arreglo, cuando la suma es mayor que la base (10) se hace acarreo y se continúa con el siguiente hasta recorrer todos los elementos de los arreglos.

[1] en idioma inglés 1 billón equivale a mil millones, así que 2×10^9 es igual a 2 billones en idioma inglés

2.2.1 Algoritmo Suma utilizando acarreos

El procedimiento anterior se encuentra en el seudocódigo del algoritmo algorithm1.

Algorithm 1 Algoritmo para suma de dos números grandes representados como arreglos de enteros

```
procedure ADD(A[], B[], C[])                          ▷ Output: C= A+B
    BASE ← 10
    for cada índice en A[i] y B[i] do
        sum ← A[i] + B[i] + carry
        if sum ≥ BASE then
            carry ← 1
            sum ← sum − BASE
        else
            carry ← 0
        end if
        C[i] = sum
    end for
end procedure
```

Dado que para realizar la suma es necesario recorrer los dos arreglos de entrada de n elementos, es claro que nuestro algoritmo tiene una complejidad lineal, es decir $T(n) = \emptyset(n)$.

2.3 Codificación de Producto con algoritmo de Karatsuba

La multiplicación mediante el método tradicional aprendido en la escuela básica es quizá la forma más conocida de multiplicación, sin embargo, la complejidad es cuadrática ($\emptyset(n^2)$) debido a que hay que hacer dos recorridos sobre los n elementos de los arreglos de entrada: uno para multiplicar cada elemento del arreglo A con cada elemento de B y otro para ir sumando los resultados de cada multiplicación, tal como se muestra en la siguiente figura figure5: [2]

```
                1  1  0  1
              × 1  0  1  1
-------------------------------------------
                1  1  0  1   (1101 times 1)
             1  1  0  1      (1101 times 1, shifted once)
          0  0  0  0         (1101 times 0, shifted twice)
+ 1  1  0  1                 (1101 times 1, shifted thrice)
-------------------------------------------
1  0  0  0  1  1  1  1   (binary 143)
```

Figure 1: Multiplicación tradicional.

Existe otro algoritmo para multiplicación de dos números el cual es más eficiente que la multiplicación tradicional: algoritmo de Karatsuba el cual es del tipo de algoritmos "Divide y Vencerás".

Se codificó la función `int karatsuba(int numX[], int numY[], int result[])` que realiza la multiplicación de `numX` y `numY` y coloca el resultado en `result` mediante el método de Karatsuba.

2.3.1 Historia

En otoño de 1960, Kolmogorov, un brillante matemático ruso, organizó un seminario acerca de problemas matemáticos de cibernética en la Universidad Estatal de Moscú, donde planteó algunos

problemas de complejidad computacional. Kolmogorov estableció que dos dígitos de n-bits no podrían ser multiplicados en menos de n^2 operaciones. En una semana, Karatsuba, un estudiante de 25 años, encontró un algoritmo "Divide y vencerás" que multiplicaba dos números de n dígitos en n^{log3} operaciones, refutando así la suposición inicial de Kolmogorov. Kolmogorov hizo público tal descrubrimiento hasta 1962, en la revista científica soviética Proceedings of the USSR Academy of Sciences. El artículo había sido escrito por Kolmogorov, en colaboración con Yuri Ofman, pero nombraba a A. Karatsuba y Yu. Ofman como los autores. Karatsuba se dio cuenta de la publicación cuando recibió una copia del artículo por parte de la editorial de la revista.

2.3.2 Algoritmo de Karatsuba

Como se ha visto arriba, el método tradicional para multiplicar 2 números de n-bits requiere n^2 multiplicaciones. El algoritmo de Karatsuba sólo requiere n^{log3} o aproximadamente $n^{1.53}$ operaciones.

Sea x y y dos números de n-bits [2], partimos a x y y en dos mitades como se muestra en la figura figure2.

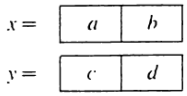

Figure 2: Partición de x y y.

Si tratamos cada mitad como un número de $\frac{n}{2}$ bits, entonces el producto lo podemos expresar como sigue

$$xy = (a2^{\frac{n}{2}} + b)(c2^{\frac{n}{2}} + d) \quad (1)$$

$$= ac2^n + (ad + bc)2^{\frac{n}{2}} + bd \quad (2)$$

La ecuación align2 calcula el producto de x y y con cuatro multiplicaciones de números de $\frac{n}{2}$-bits y algunas sumas y corrimientos (multiplicaciones por potencias de la base 2). [1, p. 62] El producto z de x y y puede ser computado por el siguiente programa:

$u \leftarrow (a + b) * (c + d)$;
$v \leftarrow (a * c)$;
$w \leftarrow (b * d)$;
$z \leftarrow v * 2^n + (u - v - w) * 2^{\frac{n}{2}} + w$;

---

el seudocódigo anterior requiere sólo 3 multiplicaciones de números de $\frac{n}{2}$-bit y algunas sumas y multiplicaciones por la propia base (corrimientos). Además, para evaluar el producto de las operaciones de u, v y w se puede invocar recursivamente al procedimiento. [1, p. 63]

Debido a que es un procedimiento recursivo y las sumas y corrimientos requieren $\emptyset(n)$, entonces la complejidad computacional de multiplicar dos números de n-bits está acotado por:

[2] por simplicidad para esta explicación asumiremos que n es potencia de 2, es decir el sistema binario, sin embargo el programa desarrollado trabaja con sistema decimal

$$T(n) = \begin{cases} k & \text{para } n = 1) \\ 3T(\frac{n}{2}) + kn & \text{para } n > 1 \end{cases} \tag{3}$$

La solución de la recurrencia align3 está acotada por arriba por:

$$3kn^{log3} \simeq 3k^{1.53}$$

Se ha visto que el algoritmo de Karatsuba reduce el número de operaciones de cuatro a tres multiplicaciones, lo cual reduce en 25% el tiempo computacional en cada nivel de ejecución que se invoca recursivamente. [1, p. 64]

La imágen muestra una comparación de tiempos del método tradicional y la multiplicación usando Karatsuba.

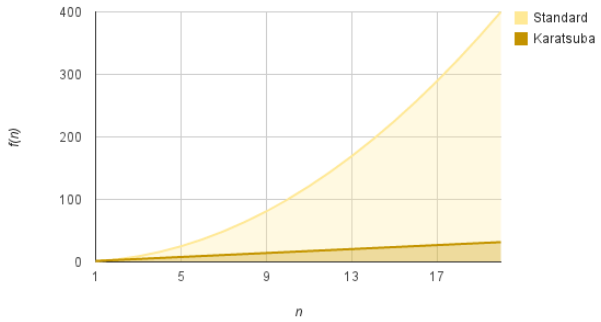

Figure 3: Tiempos computacionales de multiplicación tradicional y Karatsuba.

Se codificó la función `int karatsuba(int numX[], int numY[], int result[])` la cual implementa el algoritmo de Karatsuba (la varaible B representa la base decimal). El algoritmo algorithm2 contiene dicho seudocódigo.

2.3.3 Otras funciones codificadas

Las siguientes funciones son necesarias para el funcionamiento del programa, por ejemplo la función `toLeft` es de suma importancia ya que hace un corrimiento a la izquierda que equivale a multiplicar un número por una potencia de su base (en este caso decimal) y es un paso necesario por el algoritmo de Karatsuba.

- main: Función principal del programa. Desde aquí se invoca a la función karatsuba.
- karatsuba: Función que implementa el algoritmo de karatsuba para multiplicar dos números grandes.
- add: Implementa la suma de dos números grandes.
- substraction: Implementa la resta de dos números grandes.

Algorithm 2 Multiplicación por método de Karatsuba

procedure KARATSUBA($A[], B[], C[]$) ▷ Output: result= numX*numY
  $a \leftarrow$ mitad izquierda de numX
  $b \leftarrow$ mitad derecha de numX
  $c \leftarrow$ mitad izquierda de numY
  $d \leftarrow$ mitad derecha de numY
  $u1 \leftarrow a + b$
  $u2 \leftarrow c + d$
  $u \leftarrow$ KARATSUBA(u1, u2)
  $v \leftarrow$ KARATSUBA(a, c)
  $w \leftarrow$ KARATSUBA(b, d)
  return $v * B^n + (u - v - w) * B^{\frac{n}{2}} + w$;
end procedure

- intToArray: Convierte a arreglos de enteros un número ingresado en modo caracter.
- toLeft: Hace un corrimiento a la izquierda equivalente a multiplicar a un número por una potencia de su base.
- substraction: Implementa la resta de dos números grandes.
- incrementa: Incrementa en 1 un número grande.
- makeEvenSize: Agrega el número de ceros necesarios a un arreglo para que su longitud sea par logrando que se pueda dividir en 2 por el algoritmo de Karatsuba.
- doEqualsDigits: Convierte la longitud de dos arreglos en iguales agregando ceros.
- putsOnes: Función auxiliar que inicializa arreglos.
- readNumber: Convierte números leídos en caracter en representación de arreglos de enteros.

Los códigos de estas funciones se pueden ver en el archivo `main.c`.

3 Manual de usuario

3.1 Compilación del programa

Para la compilación del programa se debe tener instalado el compilador GCC. Luego ir a la carpeta del código fuente y escribir lo siguiente:

```
gcc main.c -o main.o
```

3.2 Ejecución

Para ejecutar el programa basta abrir una línea de comandos e ir a la carpeta del código fuente del programa, previamente compilado, y escribir lo siguiente y presionar la tecla *enter*:

```
./main.o <option>
```

donde *option* es el tipo de operación que deseamos hacer: 1 para sumar, 2 para multiplicar mediante algoritmo de Karatsuba. Por ejemplo:

Para hacer una suma: `./main.o 1`
Para multiplicar por Karatsuba `./main.o 2`

a continuación el progama solicita los 2 números a calcular, se debe ingresar cada número mediante el teclado (o copiando y pegando) y presionando la tecla *enter* tras haber ingresado cada uno. El programa calculará el resultado.
Ejemplo de ejecución (figura figure4):

```
Yanis-MacBook-Air:karatsubaNB yani$ ./main.o 1
Num. 1: 12345678912345678912345678912
```

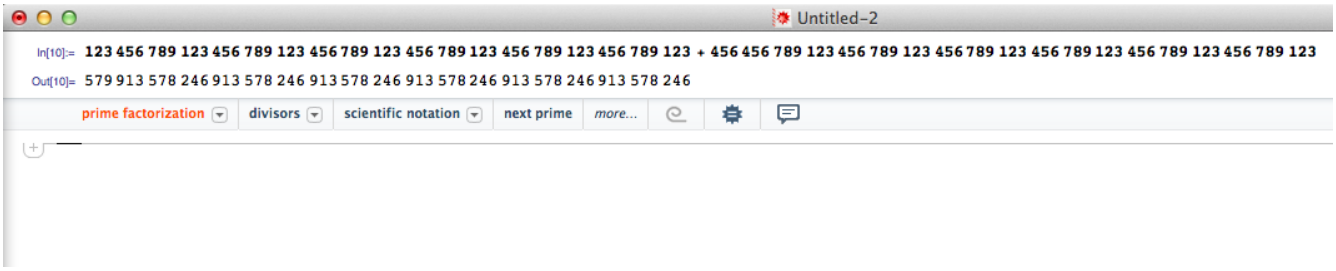

Figure 6: Comparación con Mathematica

4.2 Ejemplo 2 - Suma

Se ejecutó la suma con números grandes mayores de 100 dígitos:

```
Yanis-MacBook-Air:karatsubaNB yani$ ./main.o 1
Num. 1: 456789123456123456789123456789123456789123456789123456789123456789123456789123456789123456789123456789123

Num. 2: 935634123456123456789123456789123456789123456789123456789123456789123456789123456789123456789123456789123

RESULTADO:
1392423246912246913578246913578246913578246913578246913578246913578246913578246913578246913578246913578246
Yanis-MacBook-Air:karatsubaNB yani$
```

Figure 7: Ejecución del programa con número de más de 100 dígitos

Se compara el resultado obtenido en Mathematica y es correcto:

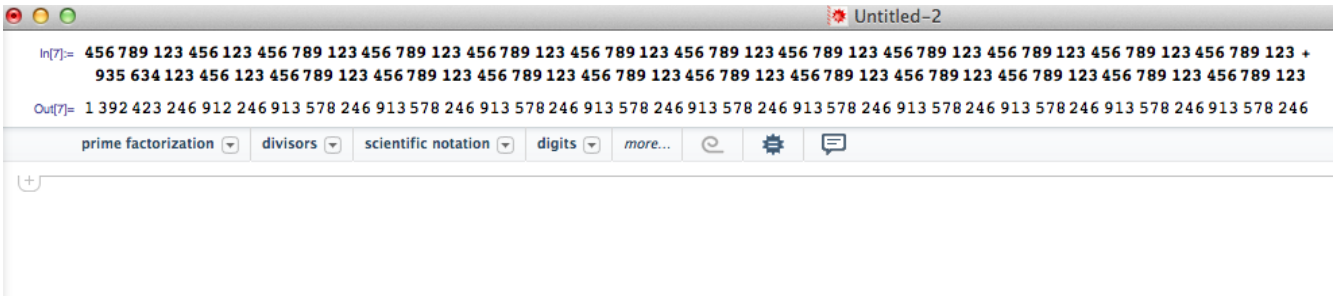

Figure 8: Comparación con Mathematica

4.3 Ejemplo 3 - Multiplicación con Karatsuba

```
Yanis-MacBook-Air:karatsubaNB yani$ ./main.o 2
Num. 1: 456789123456123456789123456789123456789123456789123456789123456789123456789123456789123456789123456789123

Num. 2: 935634123456123456789123456789123456789123456789123456789123456789123456789123456789123456789123456789123

RESULTADO:
427387491129161033487100629030311394462411159894510916818610696107342409691662597345982785000302972654623138187541484
544978899801768923258558946715348970172138993628929017085719040542509063999299087456089110912 9
Yanis-MacBook-Air:karatsubaNB yani$
```

Figure 9: Ejecución de multiplicación de Karatsuba para dígitos mayores de 100.

Se compara el resultado obtenido en Mathematica y es correcto:

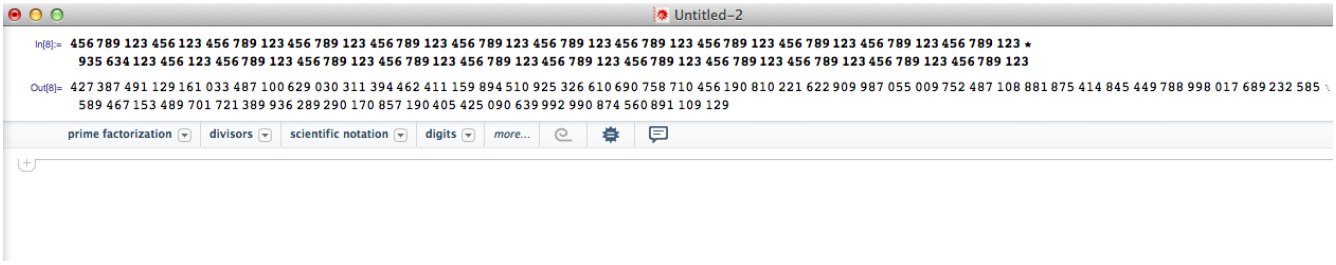

Figure 10: Comparación con Mathematica

4.4 Ejemplo 4 - Multiplicación con Karatsuba

```
Yanis-MacBook-Air:karatsubaNB yani$ ./main.o 2
Num. 1: 57444979123456123456789123456789123456789123456789123456789123456789123456789123456789123456789123456789121

Num. 2: 12894867963748695574791234567891234567891234567891234567891234567891234567891234567891234567891234567891235

RESULTADO:
740745420977266989862829335359366276426684974081005967566258048336203071104017156666560358543103560419646765080376060
287627197304332504048900404283468304518036204752604104987172005221739905456307805690875705925443 5
Yanis-MacBook-Air:karatsubaNB yani$
```

Figure 11: Ejecución de multiplicación de Karatsuba para dígitos mayores de 100.

Se compara el resultado obtenido en Mathematica y es correcto:

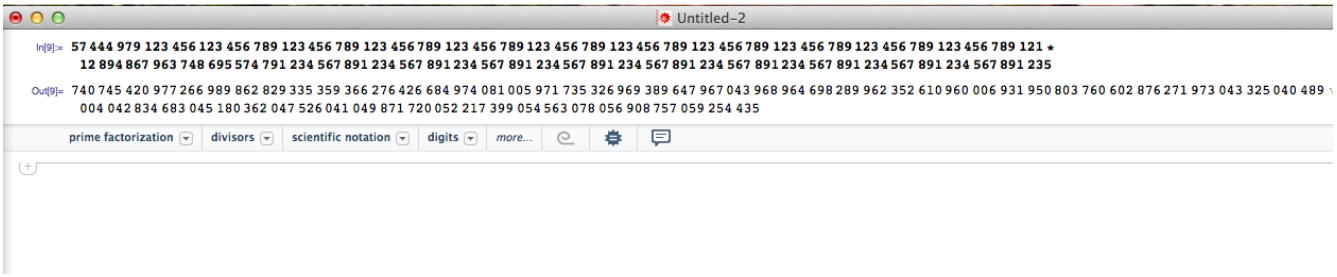

Figure 12: Comparación con Mathematica

References

[1] Ullman Aho, Hopcroft. *The Design and Analysis Of Computer Algorithms.* Addison-Wesley, 1974.

[2] C. H. Papadimitriou S. Dasgupta and U. V. Vazirani. *Algorithms.*